1)

(a) List three (03) benefits of using a layered network model in a typical protocol stack.

(6 marks)

ANSWER IN THIS BOX

Facilitates troubleshooting

Breaks complex networking into manageable layers

Allows interoperability between layers developed by different vendors.

➔ **2 mark for benefit**

(b) Explain the difference between **fault-tolerance** and **fault-resilience** in distributed computing.

(4 marks)

ANSWER IN THIS BOX

Fault-Tolerance: The system continues to operate correctly even when faults occur.
Fault-Resilience: The system recovers quickly from faults.

(c) In which ways can a distributed system be considered less advantageous than a centralised system in relation to complexity in design and maintenance?

(5 marks)

ANSWER IN THIS BOX

Distributed systems are inherently more complex to design, implement, and maintain due to the need for coordination among multiple nodes.
Centralized systems are simpler, with a single point of control and easier management.

(d) List and briefly explain three types of system failure models.

(6 marks)

ANSWER IN THIS BOX

Halting Failures: Process stops without incorrect actions.
Send-Omission Failures: Messages are not sent when they should be.
Byzantine Failures: Arbitrary or malicious behavior, such as sending incorrect or inconsistent messages.

(e) List four (04) characteristic features of a distributed system.

(4 marks)

ANSWER IN THIS BOX

Multiple autonomous hosts
Components that interact and request services from each other
Middleware to manage communication and resolve heterogeneity
Perception of a single, integrated system to the user

2)

(a) Define the term *middleware* and explain its purpose.

(5 marks)

ANSWER IN THIS BOX

Middleware is software that facilitates communication and data management for distributed applications. It connects:

- Presentation and application layers
- Application to application layers
- It is essential for building scalable, distributed systems.

(b) Can middleware be used in non-distributed systems? Explain.

(4 marks)

ANSWER IN THIS BOX

– Yes. Middleware can be used on a single machine for:

Testing distributed features
Building applications that may be distributed in the future

(c) Distinguish between an *operating system* and *middleware* in three (03) aspects.

(4 marks)

ANSWER IN THIS BOX

Operating System (OS): Provides core kernel-level functionality.
Middleware: Offers domain-specific features and sits between the OS and application layer.
Some middleware functions (e.g., TCP/IP stack) are now integrated into modern OSs.

(d) Explain the use and operation of Object-Oriented Middleware.

(5 marks)

ANSWER IN THIS BOX

Object-oriented middleware supports distributed object requests. It allows a client object to make a logical method call to a remote object, enabling interaction across distributed systems.

(e) Explain the functionality of the code below.

(7 marks)

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
import java.util.HashMap;
import java.util.Map;

public class StudentImpl extends UnicastRemoteObject implements Student {
    private Map<Integer, String> studentData;

    protected StudentImpl() throws RemoteException {
        super();
        studentData = new HashMap<>();
        studentData.put(1, "John Doe, Age: 21, Course: Computer Science");
        studentData.put(2, "Jane Smith, Age: 22, Course: Mathematics");
    }

    @Override
    public String getStudentDetails(int id) throws RemoteException {
        return studentData.getOrDefault(id, "Student not found.");
    }

    @Override
    public String addStudent(int id, String details) throws RemoteException {
        if (studentData.containsKey(id)) {
            return "Student ID already exists!";
        }
        studentData.put(id, details);
```

```
            return "Student added successfully.";
        }
    }
```

ANSWER IN THIS BOX

The class StudentImpl is a remote object that implements the Student interface (not shown here, but assumed to define the methods getStudentDetails and addStudent). It allows clients to:

Retrieve student details by ID.
Add new student records to the system.
This is part of a distributed application where clients can invoke methods on this object remotely over a network using Java RMI.

3.

(a) What are the eight (08) elements of middleware?

(08 marks)

ANSWER IN THIS BOX

Communication Link
Protocols (Network and Middleware)
Programmatic Interface
Common Data Format
Server Process Control
Naming and Directory Services
Security
Systems Management

(b) What is meant by marshalling in Remote procedure call (RPC)?

(4 marks)

ANSWER IN THIS BOX

Marshalling is the process of converting parameters into a message format suitable for transmission over a network. It ensures compatibility between different machine architectures (e.g., little-endian vs. big-endian).

(c) What is serialization and how is it related to marshalling?

(4 marks)

ANSWER IN THIS BOX

Serialization is the process of converting an object into a message for storage or transmission. It is a broader concept that includes marshalling, especially in object-oriented systems.

(d) List four (04) weaknesses of RPC.

(4 marks)

ANSWER IN THIS BOX

Lack of multithreading (client blocks during call)
Server needs multiple threads to handle clients
Synchronization issues with shared resources
Non-atomic failures in distributed systems
Performance overhead compared to local calls
Difficult debugging and testing in multithreaded environments

(e) Define a Message Broker in the context of middleware architectures.

(5 marks)

ANSWER IN THIS BOX

A message broker is an intermediary software module that translates messages between different messaging protocols used by the sender and receiver. It is a key component in message-oriented middleware and helps decouple applications.

4.

(a) What are the consequences of not having IDL in *message queueing* (message oriented middleware)?

(5 marks)

ANSWER IN THIS BOX

Message queueing lacks an Interface Definition Language (IDL), so there is no automatic marshalling. Developers must ensure both sender and receiver understand the message format, often requiring special formatting tools.

(b) Identify the most suitable **HTTP verb** to be used in each of the following RESTful URIs.

(5 marks)

ANSWER IN THIS BOX

URI	HTTP Method
/coffeOrder/{id}	GET
/coffeOrder/add	POST
/coffeOrder/delete/{id}	DELETE
/coffeOrder/getAll	GET
/coffeOrder/update/{id}	PUT

(c) The following piece of code was taken from a **Controller class** in a Restful backend application.

Explain The *functionality* of the code given below.

(10 marks)

```
package com.example.coffeeapp.controller;

import com.example.coffeeapp.model.CoffeeOrder;
```

```
import com.example.coffeeapp.service.CoffeeOrderService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/orders")
public class CoffeeOrderController {

    @Autowired
    private CoffeeOrderService coffeeOrderService;

    @PostMapping
    public ResponseEntity<CoffeeOrder> createOrder(@RequestBody CoffeeOrder
    order) {
        CoffeeOrder createdOrder = coffeeOrderService.createOrder(order);
        return ResponseEntity.ok(createdOrder);
    }

    @GetMapping
    public ResponseEntity<List<CoffeeOrder>> getAllOrders() {
        return ResponseEntity.ok(coffeeOrderService.getAllOrders());
    }


    @GetMapping("/{id}")
    public ResponseEntity<CoffeeOrder> getOrderById(@PathVariable Long id) {
        CoffeeOrder order = coffeeOrderService.getOrderById(id);
        return    order    !=    null    ?    ResponseEntity.ok(order)    :
    ResponseEntity.notFound().build();
    }


    @PutMapping("/{id}")
    public ResponseEntity<CoffeeOrder> updateOrder(@PathVariable Long id,
    @RequestBody CoffeeOrder updatedOrder) {
        CoffeeOrder order = coffeeOrderService.updateOrder(id, updatedOrder);
        return    order    !=    null    ?    ResponseEntity.ok(order)    :
    ResponseEntity.notFound().build();
    }
```

```
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteOrder(@PathVariable Long id) {
        boolean deleted = coffeeOrderService.deleteOrder(id);
        return     deleted     ?     ResponseEntity.noContent().build()     :
    ResponseEntity.notFound().build();
    }
}
```

ANSWER IN THIS BOX

The CoffeeOrderController class is a RESTful controller in a Spring Boot application that manages coffee orders. It defines endpoints for creating (POST), retrieving (GET), updating (PUT), and deleting (DELETE) coffee orders. The controller uses CoffeeOrderService to handle business logic. Each method maps to a specific HTTP request and returns a ResponseEntity with appropriate status codes and data. The controller supports operations like fetching all orders, retrieving an order by ID, updating an existing order, and deleting an order. It uses annotations like @RestController, @RequestMapping, and @Autowired to integrate with Spring's dependency injection and routing mechanisms.

(d) The following service call was found in a piece of code in a **Controller class** of a Restful backend application.

```
Order order = orderService.findOrderById(orderId);
```

Briefly explain the expected functionality of the above *service call*.

(5 marks)

ANSWER IN THIS BOX

Service classes implements the business logic (methods to create, retrieve, and manipulate data) for a particular entity. Here the route service.findOrderById method will output a order object after searching for a order using a orderID.

*****